WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Term	Documents
FUNCTIONAL.USPT.	302402
FUNCTIONALS.USPT.	114
UNITS.USPT.	517520
UNIT.USPT.	996437
(4 AND (FUNCTIONAL ADJ UNITS)).USPT.	0
(L4 AND (FUNCTIONAL ADJ UNITS)).USPT.	0

Database:
US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L5

Refine Search | Recall Text | Clear

Search History

DATE: Tuesday, October 15, 2002 Printable Copy Create Case

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
	DB=USPT; PLUR=YES; OP=ADJ		
L5	L4 and (functional adj units)	0	L5
L4	(5440714).pn. or (4926323).pn.	2	L4
L3	L2 and (global same local)	15	L3
L2	L1 and ((register adj file) with (partition$))	62	L2
L1	((712/$)!.CCLS.)	7648	L1

END OF SEARCH HISTORY

WEST

Generate Collection

L3: Entry 15 of 15 File: USPT May 15, 1990

DOCUMENT-IDENTIFIER: US 4926323 A
TITLE: Streamlined instruction processor

Detailed Description Text (100):
As illustrated in FIG. 9, register numbers 0 and 1 store special information, register numbers 64-127 are global registers, and register numbers 128-255 are local registers. Register numbers 2-63 are not implemented.

Detailed Description Text (102):
Global register 1 contains the stack pointer, which is used in the addressing of local registers as explained below.

Detailed Description Text (103):
Global registers 64-127 are accessed with the 7 least significant bits of the register number from the subject instruction. Local registers 128-255, indicated when the most significant bit of the register number is 1, are addressed by adding bits 8-2 of the stack pointer to the 8-bit register number and truncating the result to 7 bits. The most significant bit of the original register number is left at 1.

Detailed Description Text (104):
The stack pointer is a 32-bit register that may be an operand of an instruction as any other general purpose register. However, a shadow copy of global register 1 is maintained by processor hardware to be used in local register addressing. This shadow copy is set with the results of arithmetic and logic instructions. If the stack pointer is set with the result of any other instruction class, local registers cannot be accessed predictably until the stack pointer is once again set with an arithmetic or logical instruction. A modification of the stack pointer has a delayed effect on the addressing of the local registers. An instruction which writes to the stack pointer or indirect pointer can be immediately followed by an instruction which reads the stack pointer or indirect pointer. However, any instruction which references a local register also uses the value of the stack pointer or indirect pointer to calculate an absolute register number. Because of the pipeline implementation, at least one cycle of delay must separate an instruction which updates the stack pointer or indirect pointer and an instruction which references a local register. In most systems, this affects procedure call and return type instructions only. In general, though, an instruction which immediately follows a change to the stack pointer or indirect pointer should not reference a local register. Note that this restriction does not apply to a reference of a local register via an indirect pointer that has not been subject of an update.

Detailed Description Text (121):
As illustrated in FIG. 10, the general purpose registers in the register file 40 are partitioned into 16 banks, each including 16 registers; however, lower banks are unimplemented in the system described in the Am29000 User's Manual. Register positions 0 and 1 are not included in bank 0 as they are not, in fact, general purpose registers. A special purpose register in the special purpose register file 54 holds 16 register bank protection bits as indicated in the left hand column of FIG. 10. One bit corresponds to each of the 16 banks of general purpose registers. When the register protection bit for a given bank is 1, attempts accesses to that register as indicated by the register number on line 828 at the output of A multiplexer 814, line 827 at the output of B multiplexer 820, or line 824 at the output of C multiplexer 823, then a register bank protection trap is asserted by the processor. This gives the programmer the ability to restrict access to banks of registers in the register file 40. Register bank protection works only in the user mode and has no effect in the supervisor mode. Note that the protection is based on

absolute register numbers; and in the case of local registers, stack pointer addition is performed before protection checking.

Current US Original Classification (1):
712/238

Current US Cross Reference Classification (1):
712/207

CLAIMS:

7. The apparatus of claim 5, wherein the file of data locations in the storage means includes a first subset of local data locations and a second subset of global data locations, and wherein the means for generating file addresses is responsive to the stack pointer in generation of file addresses identifying locations within the first subset.

30. The apparatus of claim 5, wherein the file of data locations in the storage means includes a first subset of local data locations and a second subset of global data locations, and wherein the means for generating file addresses is responsive to the stack pointer in generation of file addresses identifying locations within the first subset.

L3: Entry 14 of 15 File: USPT Aug 8, 1995

DOCUMENT-IDENTIFIER: US 5440714 A
TITLE: Method and system configuration for simplifying the decoding system for access to an register file with overlapping windows

Abstract Text (1):
The present invention comprises a decoding system for decoding a data accessing instruction for accessing data stored in a plurality of registers wherein the registers are of different types including a global type, a local type, an input type and an output type, the registers being cataloged into a plurality of windows arranged in a predefined window sequence wherein each window including a plurality of registers of each of the types arranged in a predefined register sequence wherein the output registers of one of the windows being overlapping with the input registers of an adjacent window which being next in sequence of the window sequence. The decoding system comprises an instruction issuing means for issuing a data accessing instruction including a plurality of bits wherein the bits being encoded in an order corresponding to the window sequence and the register sequence and a set of bits of the instruction is used for defining a corresponding window and a corresponding type of the registers. The decoding system further comprises a decoding means for decoding each sets of bits of the instruction utilizing the overlapping of input registers with output registers between two adjacent register windows to select a register in one of windows for retrieving the stored data therefrom.

Brief Summary Text (6):
For the CPU of a reduced instruction set computer (RISC), a structure in the form of register files are often used for the construction of the top level memory because the data can be retrieved at a very high access rate since the register mode instructions for data retrieval are high efficiency data access instructions. FIG. 1 shows the organization of a register file 1 which is partitioned into a plurality of fixed-size, overlapping `windows`, e.g., window A (2) and window B (4), wherein each `window` provide access to the CPU (not shown) when it is `visible`. Not all registers are simultaneously accessible to the CPU at any given time. Generally, only one window is accessible, i.e., visible, and that window is denoted as `current window` (6). The current window 6 is selected by the CPU which makes the selection by generating a window number which is then decoded by a register file decoder 8 to point to the selected window and utilize that window as the current window. The CPU is meanwhile executing a plurality of instructions. A register number 10 is selected by the instructions which again is processed by the decoder 8 to select a register in the current window 6 selected by the CPU.

Brief Summary Text (7):
FIG. 1 shows that some registers belong to two different windows but have different register number in each window. Register r.sub.0 in window A is register r.sub.3 in window B. Such registers are referred to as overlapping registers. Some registers belong to only one window and they are referred to as `local` registers 12. Registers r.sub.1, r.sub.2 and r.sub.3 are local registers 12 in window A and registers r.sub.0, r.sub.1, and r.sub.2 are local registers 12 in window B. In addition, the register file structure for a RISC CPU further comprises a plurality of global registers (not shown in FIG. 1, see FIG. 2) which belongs to all windows and can be accessed at any given time by the CPU. The use of an overlapping window architecture in configuring a RSIC register file has many associated benefits that will become clear from the discussion below. More details are disclosed in `RISC I & II Architecture and Pipeline` in `Reduced Instruction Set Computer Architecture for VLSI` by Manolis G. H. Katevenis, MIT Press 1985.

Brief Summary Text (11):
FIG. 2 shows a circular stack buffer comprises register files 20 organized into eight windows, i.e., w.sub.1,w.sub.2, . . . w.sub.8. At any given time, a program can address 32 registers including eight `ins` registers, eight `locals` registers, eight `outs` registers, and eight `global` registers (as is dearly denoted in FIG. 2). The eight `global` registers are addressable from any window. The eight `outs` of one window are also the eight `ins` of the adjacent window. Although an instruction can address twenty-four windowed registers and eight global registers, excluding these global registers, a single window actually comprises sixteen registers, i.e., eight `ins` and eight `locals`. The overlapping nature of the register window can be used to pass information quickly between the overlapping `ins` and `outs` in two adjacent windows for a multi-tasking operation which is often encountered under the working environment of UNIX. There is no need to read and write these common data as they are simply shared by allowing access to the common addressable memory locations.

Brief Summary Text (21):
Briefly, in a preferred embodiment, the present invention comprises a decoding system for decoding a data accessing instruction for accessing data stored in a plurality of registers wherein the registers are of different types including a global type, a local type, an input type and an output type, the registers being cataloged into a plurality of windows arranged in a predefined window sequence wherein each window including a plurality of registers of each of the types arranged in a predefined register sequence wherein the output registers of one of the windows being overlapping with the input registers of an adjacent window which being next in sequence of the window sequence. The decoding system comprises an instruction issuing means for issuing a data accessing instruction including a plurality of bits wherein the bits being encoded in an order corresponding to the window sequence and the register sequence and a set of bits of the instruction is used for defining a corresponding window and a corresponding type of the registers. The decoding system further comprises a decoding means for decoding each sets of bits of the instruction utilizing the overlapping of input registers with output registers between two adjacent register windows to select a register in one of windows for retrieving the stored data therefrom.

Detailed Description Text (3):
An operation register of five bits is used to address the thirty-two, i.e., 32=2.sup.5, registers in each window which is divided into four different types, i.e., the `ins`, the `locals`, the `outs`, and the `global` wherein each type comprises eight registers. Table 1 shows the addressing algorithm used by the present invention for each type of registers.

Detailed Description Text (5):
Since the basic principle of operation in an overlapping window architecture is to overlap the `ins` and `outs` of adjacent windows such that the register flush requirement and the input and output operations are eliminated. Other than eight `globals` which are addressable from every window, each window has sixteen `effective` registers, i.e., eight `locals` and eight overlapped registers. FIG. 3 illustrates the concept of the effective registers wherein the effective registers which are visible to the calling procedure and CPU are the shaded areas 30 which include the `globals` 32, the `ins` 34 and the `locals` 36. For that reason, a decoding system according to the present invention only has to address the registers within the `locals` 36 and the `ins` 34. Table 1 shows the addressing scheme of such an window overlapping system. For each window, the global registers are assigned with addresses of zero to seven, the local registers with zero to seven and the `ins` registers from eight to fifteen. For every procedure call which uses the `outs` registers, the decoder automatically subtract one from the CWP and point to the overlapped registers in the `ins` register of the adjacent window.

Detailed Description Text (32):
FIG. 12 shows the functional process according to the present invention showing the tasks performed by the decoder as a function of time. For the purpose of illustration, the number of registers in each widow for the types of `ins`, `outs`, `locals`, and `globals` are assumed to be equal. The window decoder 300 receives an encoded window address via a n-bit input line 302 wherein a decoding process is

performed to determine the CWP and generate a output to activated one of the 2.sup.n output lines 304. The register decoding is now performed by an overlapping and global decoder 306 and an effective register decoder 308 wherein an encoded register address received from an RS-input line 310 is converted into a 2-bit line 312 and a (m-1)-bit line 314 for inputting address data to the overlapping and global decoder 306 and the effective register decoder 308 respectively. From the overlapping and global decoder 306, an overlapping indicator is passed via an inter-decoder line 310 to the window decoder 300 and an global indicator is passed via a global-output line 312 to a bit line strobe 314. The effective register decoder generates an output which activates one of the 2.sup.(m-1) effective register output lines 316 which again is connected to the bit line strobe 314 for further processing. The bit line strobe 314 activates one of the output lines among a globals-line 318 and 2.sup.(m+n-1) effective register lines 320 to a register access processor 322 to complete the decoding process.

Detailed Description Paragraph Table (1):

TABLE 1

TYPE	REGISTERS	ADDRESS
ins	24-31	11xxx
locals	16-23	10xxx
outs	8-15	01xxx
globals	0-7	00xxx

Current US Cross Reference Classification (2):
712/41

CLAIMS:

1. A decoding system for decoding a register access instruction including a window code of N bits for defining a current window and a register code of M bits for defining a current register, for accessing an access register among a plurality of registers wherein said registers including a global type, an input type an output type, and a local type, said registers being cataloged into a plurality of overlapping windows arranged in a predefined window sequence wherein each window including a plurality of registers of each of said types arranged in a predefined register sequence wherein said input registers of one of said windows sharing a common memory location according to an input-output correlation with said output registers of an adjacent window which being one less in sequence of said window sequence, comprising:

an overlapping and global decoding means for decoding first m bits of said register code, where m<M, for determining a register type for identifying if said current register being a global register, a local register, an output, register or an input register;

a window decoding means for decoding said window code of N bits to identify said current window, said window decoding means further employing said register type from said overlapping and global decoding means for identifying an effective access window wherein:

(i) if said register type being a global register, a local register, or an output register, said effective access window being identified the same as said current window; otherwise,

(ii) said effective access window being identified as said adjacent window with one less in said window sequence; and

a register decoding means for determining an access register in said effective access window for accessing data stored therein wherein said register decoding means decoding the remaining (M-m) bits of said register code to identify said current register, and wherein:

(i) if said register type being a global register, a local register or an output register, said register decoding means identifying said access register the same as said current register; otherwise,

(ii) if said current register is an input register, said register decoding means employing said input-output correlation to identify said access register in said effective access window, which being an adjacent window with one less in said window sequence,

whereby duplicate reference to said common memory locations between said overlapping registers in said adjacent windows may be avoided.

3. A decoding system for decoding a register access instruction for accessing an access register among a plurality of registers organized in different types including a global type, an input type, an output type, and a local type, said registers being cataloged into a plurality of overlapping windows arranged in a predefined window sequence wherein each window including a plurality of registers of said types arranged in a predefined register sequence with said input registers in one of said windows overlapping according to specific input-output correlation with said output registers in an adjacent window with one prior order in said window sequence, said register access instruction including a window code for defining a current window and a register code for defining a current register, said decoding system comprising:

an overlapping and global decoding means for decoding said register code for determining a register type and an overlapping indicator wherein said overlapping indicator indicating said register is or is not an input register;

a window decoding means which employing said register type and said overlapping indicator determined by said overlapping and global decoding means for decoding said window code to identify a current window and an effective access window wherein:

(i) if said current register is an input register, said window decoding means identifying said effective access window as said adjacent window with one less in said window sequence; otherwise,

(ii) said window decoding means identifying said effective access window the same as said current window; and

a register decoding means which employing said register type determined by said overlapping and global decoding means for decoding said register code to identify said current register and an access register wherein:

(i) if said register type is an input register, said register decoding means employing said input-output correlation between adjacent overlapping windows to identify said access register in said effective access window corresponding to said current register in said current window; otherwise

(ii) said register decoding means identifying said access register the same as said current register;

whereby said access register in said effective access window may be identified and accessed by employing said input-output correction to avoid duplicate reference to said overlapping registers between said adjacent windows.